TDDB29 Compilers and interpreters

Finite Automata

Extra slide material (see whiteboard)

Christoph Kessler, IDA,
Linköpings universitet, 2007.

Why automata models?

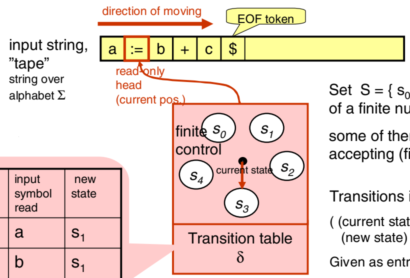
- **Automaton**: Strongly limited computation model compared to ordinary computer programs

A weak model (with many limitations) ...

- allows to do static analysis
  - e.g. on termination (decidable for finite automata)
  - which is not generally possible with a general computation model
- is easy to implement in a general-purpose programming model
  - e.g. scanner generation/coding, parser generation/coding
  - source code generation from UML statecharts
- Generally, we are interested in the *weakest* machine model (automaton model) that is still able to recognize a class of languages.

Finite Automaton / Finite State Machine

- Given by quintuple (Σ, S, s_0 in S, subset F of S, δ)



current state	input symbol read	new state
s_0	a	s_1
s_1	b	s_1
...	...	...

Set S = { s_0, s_1, ..., s_k } of a finite number of states

some of them may be accepting (final) states (F)

Transitions in δ are tuples

((current state, input symbol), (new state))

Given as entries in transition table

or as edges in a transition diagram (directed graph)

Computation of a Finite Automaton

- **Initial configuration**:
  - current state := start state s0
  - read head points to first symbol of the input string
- **1 computation step:**
  - read next input symbol, *t*
  - look up δ for entry (current state, *t*, new state) to determine new state
  - current state := new state
  - move read head forward to next symbol on tape
  - if all symbols consumed and new state is a final state: accept and halt
  - otherwise repeat

NFA and DFA

NFA (Nondeterministic Finite Automaton)

- "empty moves" (reading ε) with state change are possible, i.e. entries (s_i, ε, s_j) may exist in δ
- ambiguous state transitions are possible, i.e. entries (s_i, t, s_j) and (s_i, t, s_l) may exist in δ

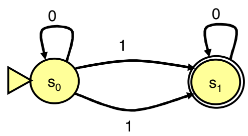
NFA **accepts** input string if there *exists* a computation (i.e., a sequence of state transitions) that leads to "accept and halt"

DFA (Deterministic Finite Automaton)

- No ε-transitions, no ambiguous transitions (δ is a function)
- Special case of a NFA

DFA Example

- DFA with
  Alphabet Σ = { 0, 1 }
  State set S = { s_0, s_1 }
  initial state: s_0
  F = { s_1 }
  δ = { (s_0, 0, s_0),
  (s_0, 1, s_1),
  (s_1, 0, s_1),
  (s_1, 1, s_0) }



- recognizes (accepts) strings containing an odd number of 1s

Computation for input string 10110:

s_0 read 1
s_1 read 0
s_1 read 1
s_0 read 1
s_1 read 0
s_1 accept

From regular expression to code

4 Steps:

- For each regular expression r there exists a NFA that accepts L_r [Thompson 1968 - see whiteboard]
- For each NFA there exists a DFA accepting the same language
- For each DFA there exists a minimal DFA (min. #states) that accepts the same language
- From a DFA, equivalent source code can be generated. [→Lecture on Scanners]

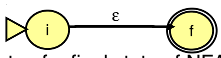
Theorem: For each regular expression r there exists an NFA that accepts L_r [Thompson 1968]

Proof: By induction,
following the inductive construction of regular expressions

Divide-and-conquer strategy to construct NFA(r):
0. if r is trivial (base case): construct NFA(r) directly, else:
1. decompose r into its constituent subexpressions r_1, r_2...
2. recursively construct NFA(r_1), NFA(r_2), ...
3. compose these to NFA(r) according to decomposition of r

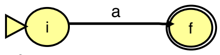
2 base cases:

Case 1: $r = \varepsilon$: NFA(r) =



with i = new start state, f = final state of NFA(r)
NFA(r) recognizes L(ε) = { ε }.

Case 2: r = a for a in Σ: NFA(r) =



recognizes L(a) = { a }.

(cont.)

4 recursive decomposition cases:

Case 3: $r = r_1 \mid r_2$: By Ind.-hyp. exist NFA(r_1), NFA(r_2)

NFA(r) =

recognizes $L(r_1 \mid r_2) = L(r_1) \cup L(r_2)$

Case 4: $r = r_1 . r_2$: By Ind.-hyp. exist NFA(r_1), NFA(r_2)

NFA(r) =

recognizes $L(r_1 . r_2) = L(r_1) . L(r_2)$

(cont.)

Case 5: $r = r_1{}^*$: By ind.-hyp. exists NFA(r_1)

NFA(r) =

recognizes $L(r_1{}^*) = (L(r_1))^*$.
(similarly for $r = r_1{}^+$)

Case 6: Parentheses: $r = (r_1)$

NFA(r) =

(no modifications).

The theorem follows by induction. □